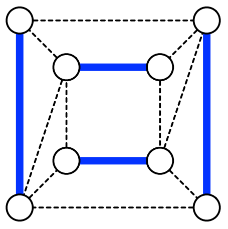
1 Matching Algorithms

Suppose we have a set of items V along with a set of allowed pairings of items $E : V \times V$, which defines a graph $G = (V, E)$. A *matching* on this graph is a subset of edges $E' \subseteq E$ such that each vertex in V participates in at most one edge in E'. Call V' the set of vertices that participate in some edge $e \in E'$, and let $M = (V', E')$ denote a matching. Depending on additional requirements, we may wish to select certain types of matchings over others. (For instance, $M = (\emptyset, \emptyset)$ is a trivial but probably useless matching in which each vertex participates in no edges.)

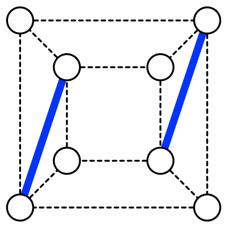
A *perfect matching* M_{perfect} is a subgraph of G such that $V' = V$, i.e., every vertex in the input graph is in the matching. This further implies that if some $(i, j) \in E'$ then no (i, k) or (j, k), for all k, can also be in the matching. That is, each vertex is paired with exactly on other vertex in the graph.

A *maximum matching* $M_{\max}$ requires that we add a weight function $w : E \to \mathbb{R}$ to our graph, and then choose a matching (not necessarily perfect) with maximum weight, i.e., the matching that maximizes $w(M) = \sum_{e \in E'} w(e)$. (If edges are unweighted, then a maximum matching of cardinality $|V|$ is also a perfect matching.)

The following illustrates these matchings. The left graph shows one of several perfect matchings (how many are there on this graph?). On the right, we have assumed a weight function that assigns the two diagonal edges weights of $> 2x$ and all other edges weight x. Note that the maximum matching is not a perfect matching, i.e., there are some vertices in V that are not in V'.



perfect matching



maximum matching

Stable matchings. Now, consider the following matching problem. Suppose that exactly n individuals are engaged in a matching process, such that when the process ends, we want all individuals to be paired with exactly one other individual. However, unlike in the perfect or maximum matching cases, in this version, each individual has its own set of *preferences* for whom they would prefer to be matched with. These preferences can be compactly represented by a vector containing a total ordering[1] of the $n - 1$ possible partners.

[1] An ordering on n items such that there are no ties and every item is included in the list.

To illustrate some properties of such a problem, consider the following small example with Alice (A), Bob (B), Carol (C) and Daniel (D), whose preference lists are the following.

person	1st	2nd	3rd
Alice	**Bob**	Carol	Daniel
Bob	Carol	Daniel	**Alice**
Carol	Alice	Bob	**Daniel**
Daniel	**Carol**	Alice	Bob

If we choose the matching (Alice, Bob) and (Carol, Daniel), then we have a problem. Alice and Daniel are happy because they are matched with their first choices, but their partners Bob and Carol would prefer to be matched with each other rather than matched with their current partners.

These pairs are called *unstable* because two individuals who are not currently matched would prefer to be matched with each other over their current partners. If a matching contains no unstable pairs and all individuals are matched, we call it a *stable matching.* In this unit, we will study the Gale-Shapley algorithm for finding such stable matchings.

1.1 Stable Marriage Problem

The version of the stable matching problem considered by Gale and Shapley (1962) is called the "stable marriage problem,"[2] and assumes two equal-sized groups of n individuals, with the goal of finding a certain kind of matching across the groups. Without loss of generality, we will call these groups "monkeys" and "walruses."

The outcome of the algorithm is a set of n stable "marriages," each composed of one monkey and one walrus, such that no pair of marriages is unstable.[3] As in the example above, each individual submits a preference list of all of their possible matches, i.e., each monkey submits a rank-ordered list of the n walruses, and each walrus submits a rank-ordered list of the n monkeys. These lists are the input to the algorithm, and the output is a stable matching on the $2n$ individuals such that each monkey is paired with exactly one walrus, and each walrus is paired with exactly one monkey.

[2]The traditional formulation and examples for the stable matching problem are strongly heteronormative, being marriages between men (m) and women (w), which reflects the problem's origination in the 1950s and 60s. These requirements can be relaxed, but doing so complicates the proofs slightly and so the heteronormative examples persist. The "monkey" and "walrus" terminology adopted here was suggested by CSCI 3104 students at the University of Colorado Boulder in Spring 2018. This terminology has several advantages.

[3]The first stable matching algorithm by Gale and Shapley was actually motivated by the college application process, in their article titled "College Admission and the Stability of Marriage." In later work, Gale and Shapley worked on a more general version of the problem, motivated by matching medical residents and hospitals. Shapley subsequently won the 2012 Nobel Memorial Prize in Economic Sciences, in part for his work on "the theory of stable allocations and the practice of market design," a direct reference to the field of algorithmic game theory, of which stable matchings is an early component.

Art by Thomas J Young, 2018

1.2 Gale-Shapley Algorithm

The key idea of the Gale-Shapley algorithm is to divide the responsibility of proposing and "disposing" among the two classes of vertices (monkeys and walruses). Specifically, monkeys (walruses) do all the proposing of matches, starting with their most preferred partners and working their way toward less preferred ones, and walruses (monkeys) accept or reject matches, working their way from less preferred to more preferred partners. The algorithm terminates when all proposals and disposals have been made. To put it succinctly, the algorithm works by having

monkeys propose and walruses dispose
or
walruses propose and monkeys dispose.

Both are equivalent versions of the Gale-Shapley algorithm, although when there is not a unique stable matching, they can yield different outcomes. We will return to this point later. Without loss of generality, we will go through the details of the monkeys-propose version.

The algorithm. To be specific, we will let each monkey begin by proposing a match with their most preferred partner. If this walrus is not yet matched, then they provisionally accepts and the pair is placed into the matching. On the other hand, if they are already matched, then they do one of two things: (i) if the monkey is lower ranked on their preference list than their current partner, then they rebuff the proposal, otherwise (ii) they accept the proposal and reject their current partner. If the monkey's proposal is unsuccessful, they repeat the process, moving down their preference list. Once a monkey is matched, we move to the next unmatched monkey and start at the next walrus down in their list (which, if the monkey was formerly matched but then became unmatched, is the walrus just lower-ranked than their previous partner.) We repeat this process until all monkeys are matched.

Without going into the data structures for storing and managing the preference lists, the pseudocode for the Gale-Shapley algorithm is straightforward. Let $m \in M$ denote a monkey and $w \in W$ denote a walrus. And, let `getWalrus(m)` be a function that returns the next highest-ranked walrus in m's list that m has not yet proposed to.

```
S = {}                              % set of pairs (m,w)
while some monkey m is free {
    w = getWalrus(m)                % choose walrus w from m's list
    if w is free { add (m,w) to S } % m,w both free, match them
    else if (m',w) in S {           % w already matched with some m'
        if m > m' in w's list {     % but w prefers m to m'
            remove (m',w) from S    %    breakup (m',w) match
            add (m,w) to S          %    create (m,w) match
            set m' as free          %    mark m as free
}}}
return S
```

This algorithm is guaranteed to halt and upon termination, it returns a stable matching—a fact that we will prove shortly. Additionally, the order in which we consider unmatched monkeys has no impact on the stable matching that is found.

Asymmetries. This version of the Gale-Shapley algorithm, in which monkeys do all the proposing, is called "monkey optimal" because when the algorithm halts, monkeys are matched with their highest-ranked walrus, conditioned on the matching being stable. (If the walruses did all the proposing, then it would be the walrus-optimal version of the algorithm.)

The monkey-optimal version, however, is not necessarily walrus-optimal, and can produce a matching that is the worse of all possible outcomes for the walruses. Consider the following set of preferences. Notice, in particular, the first column of the monkey matrix, and the last column of the walrus matrix.

monkey	1st	2nd	3rd
m_1	1	2	3
m_2	2	3	1
m_3	3	1	2

walrus	1st	2nd	3rd
w_1	2	3	1
w_2	3	1	2
w_3	1	2	3

Following the above algorithm, m_1 proposes to w_1, who accepts because they are currently free. Monkey m_2 proposes to w_2, who accepts because they are currently free. Finally, monkey m_3 proposes to w_3, who accepts, because they too are currently free. The algorithm then halts because no monkeys are free. This matching is stable because there are no unstable pairs, and each monkey is matched with their most preferred partner; however, each walrus is matched with their least-preferred partner, the worst possible stable matching from the walrus perspective.

Note that if we swapped the roles, and had walruses propose first, the resulting matching would be $\{(w_1, m_2), (w_2, m_3), (w_3, m_1)\}$. This matching is also stable, provides the walruses with their most-preferred partners, and the monkeys with their least-preferred partners. We'll revisit this possibility below.

1.3 Running time

To derive the running time of the Gale-Shapley algorithm, first consider the experience of some walrus w over the course of the algorithm. At the beginning of the algorithm, they are free and will become matched with the first monkey m who proposes to them. With each pass through the `while` loop, they may receive proposals from alternative monkeys. However, they only ever change partners if the new proposal is from a more preferred monkey in their preference list. Thus, the following property is true: a walrus w remains matched from the time they receive their first proposal until the end of the algorithm, and the sequence of their partners is a monotonically increasing sequence on their preference list.

Now consider the experience of a monkey m over the course of the algorithm. Unlike the walrsues, monkeys may become free again, in the event that their current partner receives a proposal from a more attractive monkey than them. However, because m begins by proposing to their most-preferred partner and works their way down their list, the sequence of their partners is a monotonically decreasing sequence on their preference list.

Given these insights, we can now prove that the algorithm terminates after at most n^2 iterations.

First, observe that no monkey can be rejected by all walruses. Assume that some monkey m has been rejected by all n walruses. Under the algorithm a free walrus will not reject a monkey's proposal, i.e., only a matched walrus can reject a monkey's proposal. Thus, if m has been rejected by all n walruses, then all n walruses must be already matched. However, a walrus can only be matched to at most one monkey, implying that if m is free, then at most $n-1$ walruses are matched. Thus, at least one w must still be free and m cannot have been rejected by all n walruses.

Second, each iteration of the `while` loop involves exactly one proposal. Note that because monkeys move monotonically down their preference lists, no monkey will propose to the same walrus twice. Because no monkey can be rejected by every walrus, in the worst case, a monkey will propose to every walrus before becoming matched. Thus, the number of iterations of the `while` loop is at most n^2 before the algorithm halts, and when it halts, every monkey and every walrus is matched.

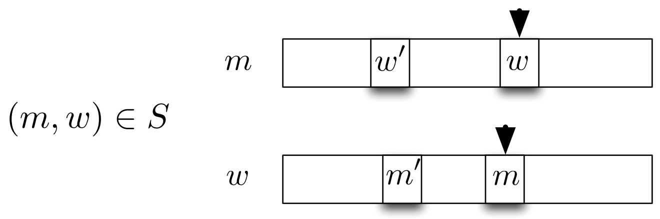
(At home exercise: what preference lists will generate the worst-case running time?)

1.4 Correctness

We now know the Gale-Shapley algorithm will halt. But it remains to be shown that it also produces a stable matching on every possible set of preferences, i.e., it is correct. Let S denote the matching produced by the Gale-Shapley algorithm.

Claim: The S produced by Gale-Shapley is always a stable matching.

Proof: Because the number of walruses and monkeys are both n, when the algorithm halts, S is a perfect matching. Assume that within S there exists an unstable pair, denoted (m, w) and (m', w') such that for m, $w' > w$ and for w', $m > m'$ (see figure below). That is, m and w' would prefer each other over their current partners. (Note that the preferences of w and m' are irrelevant.)



Over the course of the algorithm, the last successful proposal by m must have been to w. This implies two possibilities for m and w'.

- First, if m did not propose to w' before proposing to w, then w must occur higher in m's preference list than w', which contradicts our assumption that for m, $w' > w$.
- Second, if m did propose to w' at some point before being matched with w, then m was rejected by w', who preferred their current partner m'' over m. Because w' was ultimately matched with m', either $m' = m''$ or $m' > m''$. The latter case further implies that $m' > m$ because w' rejected m in favor of m''.

Thus, w' cannot prefer m to their partner m', and S must be a stable matching. $\square$

1.5 Implementing the algorithm

The preference lists themselves may be stored in two $n \times n$ matrices M and W. For the monkeys, we let $M_{i,j}$ give the jth most preferred walrus for monkey i, i.e., each row contains a monkey's list, in order of preference. For the walruses, let $W_{i.j}$ give the rank of monkey j in walrus i's list. Note that these two matrices are storing the same information, but in a slightly different way.

If monkeys are proposing, we must also store the index of the most recently proposed-to walrus, which we may do in an array v of length n. Initially, we set all $v_i = 0$ to denote the fact that no monkeys have proposed to any walruses. At some intermediate pass through the `while` loop, $v_i = j$ denotes that if i becomes free, the next walrus that i will propose to is $M_{i,j+1} = w'$. We must also store the matching itself; which we do using an adjacency list, one for the monkeys and one for the walruses.

Now, consider the pseudo-code on page 4 and ignore for the moment that we have not yet specified how we store and maintain the set of free monkeys. Given the above data structures, the function `getWalrus(m)` takes time $O(1)$ because $v_m + 1$ is the index of the next-most-preferred walrus in m's preference list, whom m has not already proposed to.

Let $w = M_{m,v_m+1}$, the next walrus to whom m will propose. If w is free, then the adjacency list containing the walruses' partners will be empty for w, which takes $O(1)$ time determine. Adding the pair (m, w) to S takes $O(1)$ time to update these two adjacency lists.

Otherwise, if w is already matched, then m' is stored in w's adjacency list, and it remains only to determine whether $m > m'$ in w's list. This can be checked in $O(1)$ time by asking $W_{w,m} > W_{w,m'}$. If m is more preferred, removing the pair (m', w) from S and adding the pair (m, w) to S can be done in $O(1)$ time. Thus, all of the operations within the `while` loop take constant time.

1.6 S is independent of the order of execution

A nice property of the Gale-Shapley algorithm is that the stable matching S it produces is independent of the order in which monkeys make their proposals. This property provides us with great flexibility in how we choose which free monkey to attempt to match next, and allows us to use a data structure that lets us choose a new free monkey in $O(1)$ time by simply storing the free monkeys in a linked list and choosing a new free monkey either from the front or back of the list. Implementing the algorithm this way provides an overall running time of $O(n^2)$.[4]

We now prove that the outcome of the algorithm is independent of the order of proposals. As before, the arguments are symmetric for monkeys-as-proposers or walruses-as-proposers. Thus, without loss of generality, we will let the monkeys do the proposing. To begin, some definitions:

Stable partner: some w for m such that there exists some stable matching S on all monkeys and walruses with $(m, w) \in S$.

[4]The best case running time is $O(n)$, in which each monkey is matched with a walrus after a constant number of proposals. The average case, however, is not $O(n^2)$ but is instead $O(n \log n)$, implying that for most preference lists, a stable matching is found fairly quickly. A nice treatment of the average case can be found here: `http://bit.ly/15c3AXt` .

Best stable partner: some w for m, such that w is a stable partner and there exists no better walrus $w' > w$ for m who is also a stable partner for m.

Claim: All possible orderings on proposals in the Gale-Shapley algorithm yield the same stable matching S and this matching is optimal for the proposers.

Proof: Assume there exists an ordering σ that produces a matching S in which a monkey m is matched with a walrus w (i.e., $(m, w) \in S$), and assume that there exists some other stable matching S' in which m is paired with another walrus w' (i.e., $(m, w') \in S'$), and furthermore that for m, $w' > w$. That is, both w and w' are stable partners for m, but w' is m's best stable partner.

Monkeys propose to walruses in decreasing order of their preferences. Thus, over the sequence σ, to arrive at S, w' must have rejected m because w' would have been proposed to by m before proposing to w. Let this be the first such rejection for m during σ, and suppose that this rejection occurred because w' was already matched with some m'. Thus, for w', $m' > m$.

By symmetry, because this was the first time a monkey has been rejected by a stable partner, m' cannot have a stable partner that is preferred to w'. Thus, for m', w' is preferred to their partner in M', and the matching M' is not stable because m' and w' would leave their current partners to be with each other (they are a "blocking pair").

Thus, each monkey m is matched in M with their best stable partner, and because σ was chosen arbitrarily, all possible executions must lead to the same stable matching M. □

1.7 Related problems

The stable matching (marriage) problem is closely related to a number of other problems, which can be viewed as variants on the stable matching problem.

- *Stable roommates*: this problem is almost identical to the stable marriage problem, except that all individuals are in a single pool.
- *Hospitals and residents* (or *college admissions*): in this version, hospital (college) i can be matched with k_i residents (applicants); the stability criterion is still enforced. As with the marriage version, the algorithms can be hospital-optimal or resident-optimal.[5]
- Sets with unequal size: instead of requiring that the number of walruses and monkeys be equal, we allow for unequal set sizes. Results similar to the Gale-Shapley algorithm apply.

[5]The matching algorithm proposed by Gale and Shapley for this problem was resident optimal, while the version used by the National Resident Matching Program, the official market clearing mechanism for matching medical residents with hospitals, is hospital optimal.